Query Rewrite Optimization for NebulaStream

Daniel Dronov
Riccardo Marin
Tobias Engelbrecht

TU Berlin

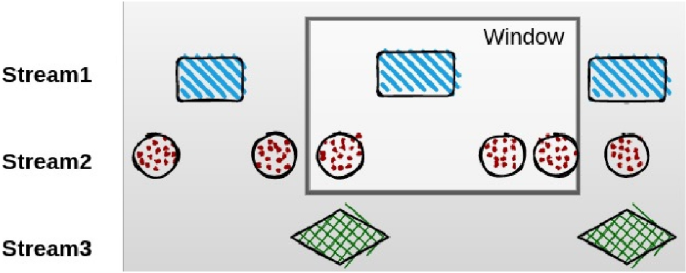
05 June 2023

Agenda

1. Background
   - Stream processing systems
   - Query plans
   - Project goal
2. Solution approach
   - Re-write rules
3. Timeline

Stream processing systems

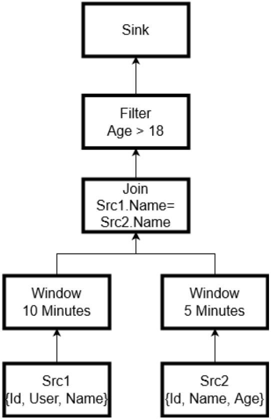
- Different sources produce data streams with well-defined schemas
- Windows are used to discretize data from the stream
- SQL-like queries are used to process the data
  - filter, map, projection, join, union, aggregations, **window**



- *NebulaStream* is a stream processing system designed the IoT

Query plans

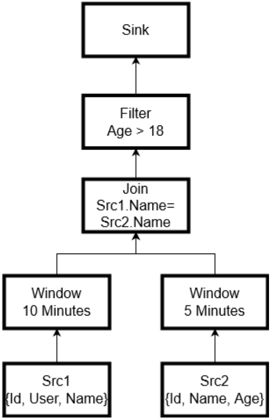
- Logical Query Plans represent queries.



Input query

Query plans

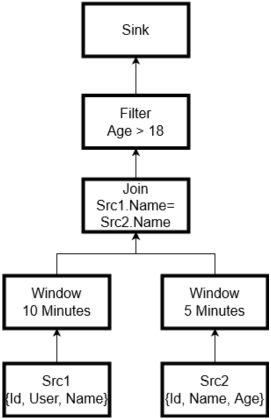
- Logical Query Plans represent queries.
- Optimizer rewrites the query



Input query

Query plans

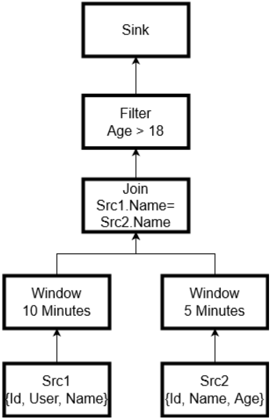
- Logical Query Plans represent queries.
- Optimizer rewrites the query
- → reduce intermediate results for each processing step



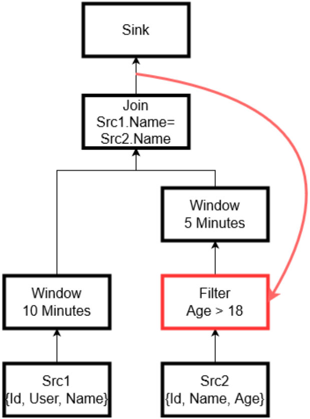
Input query

Query plans

- Logical Query Plans represent queries.
- Optimizer rewrites the query
- → reduce intermediate results for each processing step



Input query



Optimized query

Project goal

Project goal

How can query re-write rules be applied to the query plans?
What benefits and limits do they entail?

Solution approach

- We will add implementation steps in the query re-write phase
- We will verify the correctness of our assumptions with unit tests
- We will measure the performance benefits with benchmarks

Filter push-down below projection

- Can boost performance, as other operators have to iterate over less tuples
- The amount of tuples is more important than the memory size of the data

Full Table

ID	Name
1	Daniel
2	Tobias
3	Riccardo
4	Tim

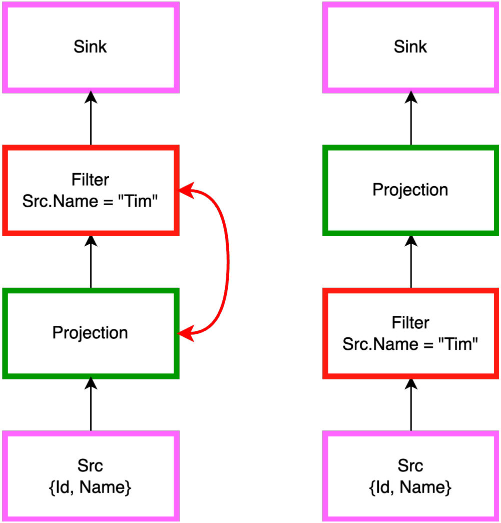
After Filtering

ID	Name
1	Daniel

After Projection

Name
Daniel

Filter push-down below projection example 1



Sink

Filter
Src.Name = "Tim"

Projection

Src
{Id, Name}

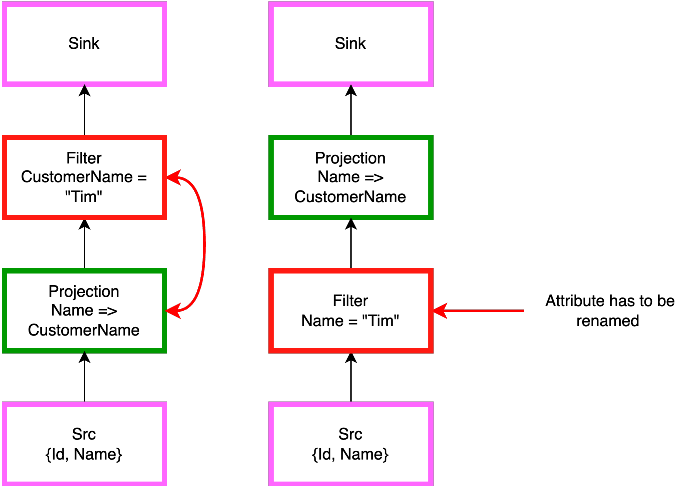
Sink

Projection

Filter
Src.Name = "Tim"

Src
{Id, Name}

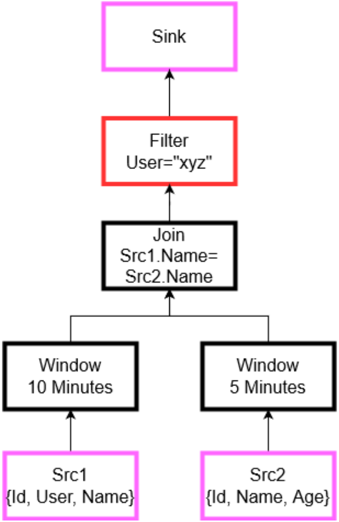
Filter push-down below projection example 2



- In case of an attribute rename, the original attribute name has to be determined

Filter push-down below join

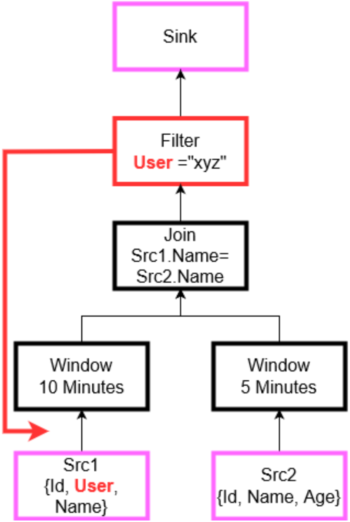
- Pushing a filter below a join reduces the intermediate results to join on
- Joins need windows in the context of streams
- Filters work on tuples, so they can be pushed below the windows
- There are three base cases



Filter and Join example

Filter push-down below join

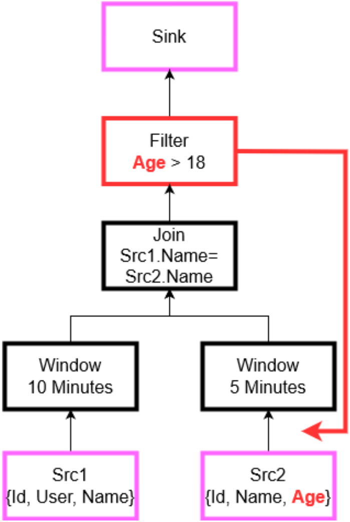
- Pushing a filter below a join reduces the intermediate results to join on
- Joins need windows in the context of streams
- Filters work on tuples, so they can be pushed below the windows
- There are three base cases
    - First case: push-down to left branch



Filter and Join example

Filter push-down below join

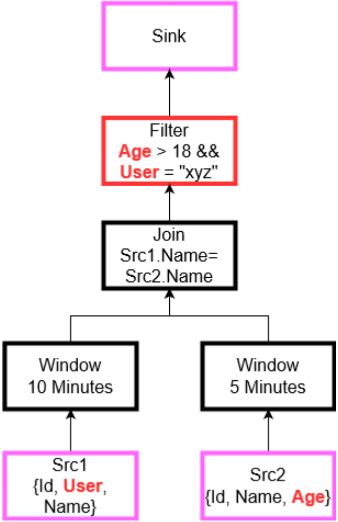
- Pushing a filter below a join reduces the intermediate results to join on
- Joins need windows in the context of streams
- Filters work on tuples, so they can be pushed below the windows
- There are three base cases
    - Second case: push-down to right side



Filter and Join example

Filter push-down below join

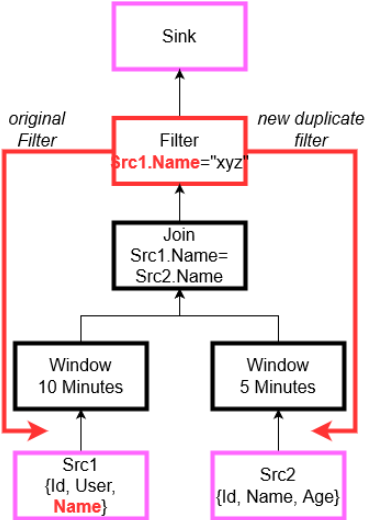
- Pushing a filter below a join reduces the intermediate results to join on
- Joins need windows in the context of streams
- Filters work on tuples, so they can be pushed below the windows
- There are three base cases
    - Third case: can't push-down



Filter and Join example

Filter push-down below join

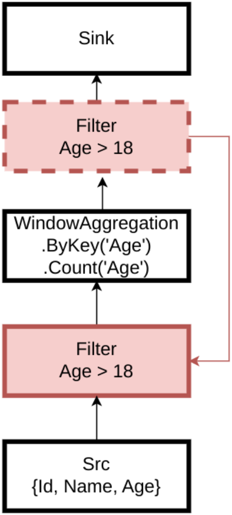
- Pushing a filter below a join reduces the intermediate results to join on
- Joins need windows in the context of streams
- Filters work on tuples, so they can be pushed below the windows
- There are three base cases
    - special case: predicate part of join-condition



Filter and Join example

Filter push-down below window aggregation

- In Nebulastream, usages for windows: aggregations and joins
- For joins, the push-down below join rule applies
- For aggregations, we can push-down if it does not impact the aggregation result
    - If there is a group by clause and filter on the same attribute, it can be pushed down



Push-down constraints

- Pushing down filters has a computational cost to take into account
- Might not be worth when the filter selectivity is low
  (= the filter is not reducing much the number of returned tuples)
- Therefore we consider adding a new step:
  1. After retrieving the filter operators
  2. Check individually their selectivity
  3. **Only if the selectivity is above a certain threshold**, push it down
- Moreover, we need to consider the operator fusion done in the compiling phase
- The point is open for discussion and will be investigated with the benchmarks

Filter predicate split up

This rule handles filter predicates with conjunctions ("and").

- The conjunctions are converted into consecutive FilterOperators
- This can potentially allow pushing one predicate below the join
- With the disjunctions we cannot apply any optimization

Filter reordering

- Identify consecutive filters
- The filters are sorted by selectivity
- → the predicates with an high selectivity are executed first
- In addition, the query plan received by the compiler is already optimal:

```
    if (p1 && p2) // Selectivity p1 >> p2
```

Projection push-down

- Filter push-down strategy can be generalized to support other operators
- In the case of projection, we can use it to reduce the number of columns passed
- Therefore we expect a performance benefit

Units tests for the rules

- Each rule will have a set of related unit tests
- In each of them we will include several variations of the query plan
  - Push-down applicable and beneficial
  - Push-down applicable but not beneficial
  - Push-down not applicable

Benchmarking

- Usage of embedded E2E Benchmark framework
- Write queries for every implemented rule
- Benchmark each optimization individually
- Analyze overall performance and improvements

Documentation and report

- C++ method documentation
- Document implemented rule
- Document e2e benchmark results and findings
- Final report including the answer to the research question

Timeline

Task	Status	When
Read and understood re-write rules	Done	01.05 - 15.05
Compared the rules with other systems	Done	07.05 - 15.05
Selected a subset of rules	Done	15.05 - 21.05
Set up of environment for Nebulastream	Done	07.05 - 15.05
Created first PR: refactoring filter push-down	Done	21.05 - 29.05
Created issues for filter push-down rules	Done	29.05 - 31.05
Filter push-down below join	In progress	22.05 - 12.06
Filter push-down below projection	In progress	22.05 - 12.06
Filter push-down below window	In progress	22.05 - 12.06
Push down constraints	In progress	22.05 - 12.06
Units tests for the rules	In progress	22.05 - 12.06
Duplicate filter if push-down below join	Planned	12.06 - 30.06
Filter predicate split up	Planned	12.06 - 30.06
Filter reordering	Planned	12.06 - 30.06
Redundancy elimination	Planned	12.06 - 30.06
Projection push-down	Planned	12.06 - 30.06
Benchmarking	Planned	01.07 - 14.07
Documentation and report	Planned	18.07 - 14.08